Enumeration of monadic second-order queries on trees

Wojciech Kazana, Luc Segoufin

Enumeration of monadic second-order queries on trees

WOJCIECH KAZANA and LUC SEGOUFIN
INRIA and ENS Cachan

We consider the enumeration problem of monadic second-order (MSO) queries with first-order free variables over trees. In [Bagan 2006] it was shown that this problem is in CONSTANT-DELAY$_{lin}$. An enumeration problem belongs to CONSTANT-DELAY$_{lin}$ if for an input structure of size n it can be solved by:

—an $O(n)$ precomputation phase building an index structure,
—followed by a phase enumerating the answers with no repetition and a constant delay between two consecutive outputs.

In this article we give a different proof of this result based on the deterministic factorization forest decomposition theorem of Colcombet [Colcombet 2007].

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms: Logic, algorithmic

Additional Key Words and Phrases: Monadic Second-Order, bounded tree-width, enumeration

1. INTRODUCTION.

Model checking is the problem of testing whether a given sentence is true in a given model. It is a classical problem in many areas of computer science, in particular in verification. When the formula is no longer a sentence but has free variables, then we are faced with the query evaluation problem. In this case the goal is to compute all the answers of a given query on a given database.

As for model checking, query evaluation is a problem often requiring a time at least exponential in the size of the query. Even worse, the evaluation often requires a time of the form $n^{O(k)}$, where n is the size of the database and k the size of the query. This is dramatic, even for small k, when the database is huge.

However, there are restrictions on the structures that make things easier. For instance a first-order (FO) sentence can be tested in time linear in n on structures of bounded degree [Seese 1996]. Here and in the sequel, the constant factors depend on the formula (in this case it is triply exponential in the size of the formula). Similarly, still over structures of bounded degree, an FO query can be evaluated in time linear in $n + m$, where m is the size of the output of the query [Frick and Grohe 2004]. Note that if the query has r variables, m could be up to n^r, in particular exponential in r, and hence in the size of the formula. Another example of particular interest for this paper is that MSO sentences can be tested in time linear in n over structures of bounded tree-width [Courcelle 1990] and MSO queries can be evaluated in time linear in $n + m$, where m is the size of the output of the query [Flum et al. 2002].

As the size m of the output may be large (possibly exponential in the size of the query), in many

Author's address: W. Kazana, kazana@lsv.ens-cachan.fr
This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513. http://webdam.inria.fr/
Author's address: L. Segoufin, see http://pages.saclay.inria.fr/luc.segoufin/
We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

applications enumerating all the answers may already consume too many of the allowed resources. In this case it may be appropriate to first output a small subset of the answers and then, on demand, output a subsequent small number of answers and so on until all possible answers have been exhausted. To make this even more attractive it is preferable to be able to minimize the time necessary to output the first answers and, from a given set of answers, also minimize the time necessary to output the next set of answers - this second time interval is known as the *delay*.

We say that a query can be evaluated in linear time and constant delay if there exists an algorithm consisting of a preprocessing phase taking time linear in n, which is then followed by an output phase printing the answers one by one, with no repetition and with a constant delay between each output. Notice that if a linear time and constant delay algorithm exists, then the time needed for the total query evaluation problem is bounded by $f(k)(n+m)$ for some function f. It was shown by Bagan in [Bagan 2006] that a linear time and constant delay query evaluation algorithm could be obtained for MSO queries over structures of bounded tree-width. A constant delay algorithm was independently obtained by Courcelle in [Courcelle 2009] but with an $O(n \log n)$ precomputation time.

Both the proofs of [Bagan 2006; Courcelle 2009] can be decomposed into two distinct steps. The first step, and most difficult one, shows that MSO queries can be evaluated with constant delay over trees. The general result for graphs of bounded tree-width then follows easily as structures of bounded tree-width can be interpreted over trees using MSO queries.

In this paper we only revisit the first step and provide an alternative proof of the existence of a linear time and constant delay query evaluation algorithm for MSO queries over trees. It is well known that, over trees, one can associate to any MSO query a tree automaton recognizing the set of trees where one tuple of a solution is marked with distinguished colors. The proof of Bagan constructs from this automaton, in linear time, an intricate index structure, which is later used by an enumeration algorithm.

In this paper we provide an alternative proof based on a deterministic factorization forest theorem proved by Colcombet [Colcombet 2007]. This result shows that, over trees, any MSO query is, modulo a recoloring of the tree, essentially a $\Sigma_2(<)$ query. As the recoloring can be done in linear time, it is therefore sufficient to provide an enumeration algorithm for $\Sigma_2(<)$ queries. These queries being fairly simple, we achieve this by induction on the structure of the formula.

Related work. Other linear time and constant delay algorithms were obtained for evaluating FO queries of structures of bounded degree [Durand and Grandjean 2007; Kazana and Segoufin 2011]. Those are either based on the locality property of FO or on a quantifier elimination technique relying on the bounded degree of the structure. In both cases it cannot be reused in our context and the techniques used here are completely different.

We rely on a deterministic factorization forest theorem proved in [Colcombet 2007]. This result states that for each MSO query, there exists a tree-shaped index structure, whose depth does not depend on the input tree, and which permits to test in constant time whether a tuple of nodes of the input tree is in the output of the query. The consequence of this result, stated here in Theorem 3.2, is explicitly stated in Theorem 2 in [Colcombet 2007] with an arbitrary FO prefix instead of a Σ_2 prefix. But the proof actually yields a Σ_2 formula and we therefore attribute Theorem 3.2 to [Colcombet 2007].

2. PRELIMINARIES.

2.1 Trees and MSO logic

We work with finite trees whose nodes are labeled using a finite alphabet. More formally, let $\mathbb{A}$ be a finite alphabet. Trees over $\mathbb{A}$ are generated by the following rules: for all $a \in \mathbb{A}$, a is a tree, furthermore if $a \in \mathbb{A}$ and, for some $k \geq 1$, $t_1, \cdots, t_k$ are trees, then $a(t_1 + \cdots + t_k)$ is a tree. We use standard terminology for trees defining nodes, root, leaves, ancestors and descendants. A *binary tree* is a tree whose every node has either no child (is a leaf) or exactly two children, the *left* child and the *right* child. Given two nodes u, v of a tree T we write $u < v$ to denote the fact that u is a strict ancestor of v. Over binary trees, we also use $u <_l v$ (resp. $u <_r v$) to express the fact that v is a descendant of the left (resp. right) child of u. Given a tree T, we denote by $|T|$ the number

of nodes of T.

As usual, we view each tree as a relational structure whose domain is its set of nodes. The signature contains a binary symbol E denoting the child relation and a unary symbol P_a per $a \in \mathbb{A}$ denoting the label of each node. For binary trees, E is replaced with two binary symbols E_1 and E_2 denoting respectively the left child and the right child relation. We consider monadic second-order logic (MSO) over these signatures allowing quantification over nodes or sets of nodes of the tree. We will also often use the binary predicates $<$, $<_l$ and $<_r$ denoting the ancestor relationships. Recall that $<$, $<_l$ and $<_r$ are definable in MSO from E or E_1 and E_2.

By *query* we mean an MSO formula whose free variables are all first order variables: we disallow free set variables. For any $\phi \in$ MSO, we denote its size, i.e. the number of symbols occurring in its syntactic tree, by $|\phi|$. When writing $\phi(\bar{x})$ we always mean that $\bar{x}$ are exactly the free (first-order) variables of ϕ. A *unary* query is a formula with exactly one free variable, i.e. of the form $\phi(x)$. Given a tree T and a tuple $\bar{u}$ of nodes of T, we write $T \models \phi(\bar{u})$ if the formula ϕ is true in T when interpreting its free variables as $\bar{u}$. Each query ϕ then defines a relation over the set of nodes of T denoted by $\phi(T)$ and whose cardinality is denoted by $|\phi(T)|$. Notice that $|\phi(T)|$ can be exponential in the number of free variables of ϕ.

2.2 Model of computation and CONSTANT-DELAY$_{lin}$ class.

We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation. For further details on this model and its use in logic see [Durand and Grandjean 2007].

The following result has been proved by Flum, Frick and Grohe. It is a generalization of the well known fact that any MSO sentence can be translated into a tree automaton and can therefore be evaluated in time linear in the size of the input tree. Note that we only mention in this paper *data complexities*, i.e. all hidden constants may depend on the size of the formula, sometime in a dramatic way, like in the result below where the constant factor is non-elementary in the size of the formula.

THEOREM 2.1. *[Flum et al. 2002] Let $\phi(\bar{x})$ be an* MSO *query. Given a tree T, $\phi(T)$ can be computed in time $O(|T| + |\phi(T)|)$.*

We will use this result only in the simple case when $|\phi(T)| = O(|T|)$. When this happens, $\phi(T)$ can be computed in time $O(|T|)$. We fall in this case when ϕ is unary or ϕ is of the form $\phi(x, y)$, but y happens to be a function of x. Notice that the fact that a binary query is the graph of a function is not in general apparent when looking at the syntax of the formula, but will be in the specific cases we will consider.

COROLLARY 2.2. *Let ϕ be an* MSO *query either of the form $\phi(x)$ or of the form $\phi(x, y)$ with y a function of x. Given a tree T, $\phi(T)$ can be computed in time $O(|T|)$.*

An enumeration problem is a binary relation. Given an enumeration problem R and an input x, a *solution for* x is a y such that $(x, y) \in R$. An enumeration problem R induces a computational problem as follows: Given an input x, output all its solutions. An enumeration problem is in the class CONSTANT-DELAY$_{lin}$ if its computational problem can be solved by a RAM algorithm which, on input x, can be decomposed into two steps:

—a precomputation phase that is performed in time $O(|x|)$,

—an enumeration phase that outputs all the solutions for x with no repetition and a constant delay between two consecutive outputs. The enumeration phase has full access to the output of the precomputation phase but can use only a constant total amount of extra memory.

In particular, if R is in CONSTANT-DELAY$_{lin}$, then the enumeration problem R can be solved in time $O(|x| + |\{y : R(x, y)\}|)$. From the best of our knowledge it is not known whether the converse is true or not. We conjecture that it is not. More details about CONSTANT-DELAY$_{lin}$ can be found in [Durand and Grandjean 2007].

We are interested in the following enumeration problem for $\phi(\bar{x}) \in$ MSO:

$$\text{Enum}(\phi) = \{(x, y) \ : \ x \text{ is a tree } T, y \text{ is a tuple } \bar{u} \text{ of nodes of } T \text{ and } T \models \phi(\bar{u})\}$$

We show that Enum(ϕ) is in Constant-Delay$_{lin}$. In other words, for a query ϕ, given a tree T we enumerate the solutions $\phi(T)$ in linear time and constant delay.

Theorem 2.3. *[Bagan 2006] For all queries $\phi \in$ MSO, Enum(ϕ) is in* Constant-Delay$_{lin}$.

The proof of Bagan computes an intricate index structure based on the tree automaton built from the MSO formula. Our proof builds on the deterministic factorization forest decomposition theorem of Colcombet [Colcombet 2007]: In Section 3 we use the deterministic factorization forest decomposition theorem of Colcombet to show that it is enough to consider first-order queries with one alternation of quantifiers! In Section 4 we show how to enumerate those queries in linear time and constant delay.

3. SIMPLIFYING THE PROBLEM.

In this section we show that it suffices to prove Theorem 2.3 for simple first-order queries using only one quantifier alternation. This reduction is obtained in several steps. All our reductions are effective, fairly simple and standard except for one, making use of a deep result on MSO queries over trees. Each step of the reduction consists in a transformation of the input tree that can be performed in linear time, together with an effective transformation of the input formula whose complexity may be non elementary. As we focus only on the complexity in the size of the input tree, the effectiveness of the transformation of the formula is only implicit in all the following statements. Therefore it can be achieved during the preprocessing phase, before the enumeration starts.

3.1 It is enough to consider binary trees.

We make use of the classical first child – next sibling encoding of unranked trees. It is folklore that this transformation can be done in time linear in the input tree and that it induces a bijection f between the nodes of the tree T and the nodes of its first child – next sibling encoding $f(T)$. Moreover, the transformation preserves MSO definability: for any MSO query $\phi(\bar{x})$ there is an MSO query $\phi'(\bar{x})$ such that for all trees T we have $f(\phi(T)) = \phi'(f(T))$. Hence, if we can enumerate $\phi'(f(T))$ in Constant-Delay$_{lin}$, we can also enumerate $\phi(T)$ in Constant-Delay$_{lin}$.

3.2 It is enough to consider queries that also output all least common ancestors.

Given a tree T and two nodes u, u' of T, the least common ancestor of u and u', denoted $\text{lca}(u, u')$, is the node v such that v is an ancestor of both u and u' and no strict descendant of v has this property. Note that there is an MSO formula defining the property $z = \text{lca}(x, y)$. An MSO query $\phi(\bar{x})$ is *lca-complete* if for all variables $x_i, x_j \in \bar{x}$ there is a variable $x_k \in \bar{x}$ such that for all T and all $\bar{u} \in \phi(T)$, $u_k = \text{lca}(u_i, u_j)$. Given an MSO query $\phi(\bar{x})$ we can compute an lca-complete MSO query $\psi(\bar{x}\bar{y})$ such that for all trees T and tuples $\bar{a} \in \phi(T)$ there is a unique $\bar{b}$, containing all the desired least common ancestors, such that $\bar{a}\bar{b} \in \psi(T)$. Typically ψ is constructed from ϕ by introducing a new free variable per pair of initial free variables, while adding the formula axiomatizing the fact that the new variable is the least common ancestor of the other two variables. For simplicity of the exposition we also assume that there is a free variable in ψ always denoting the root of the tree. Note that for all trees T we have $|\psi(T)| = |\phi(T)|$ and that all solutions in $\phi(T)$ can be reconstructed from a solution in $\psi(T)$ by projecting out the newly added component. Hence, if Enum(ψ) is in Constant-Delay$_{lin}$, we also have Enum(ϕ) in Constant-Delay$_{lin}$.

3.3 It is enough to consider queries with ancestor-typed outputs.

We now make sure that two elements of a solution are always related in a same way in all solutions. An *ancestor-type* $o(\bar{x})$ of a set of variables $\bar{x}$ is a maximal consistent conjunction of formulas of the form $x_i = x_j$, $x_i <_l x_j$, $x_i <_r x_j$ or $\neg(x_i < x_j)$. For a fixed $\bar{x}$ there are only finitely many ancestor-types that we denote by $O(\bar{x})$. For an MSO query $\phi(\bar{x})$ it is easy to see that $\phi(\bar{x})$ is equivalent to $\bigvee_{o \in O(\bar{x})} o(\bar{x}) \wedge \phi(\bar{x})$ and that for two different $o_1, o_2 \in O(\bar{x})$ and any tree T we have

$(o_1 \wedge \phi)(T) \cap (o_2 \wedge \phi)(T) = \emptyset$. Hence, if we can enumerate in CONSTANT-DELAY$_{lin}$ each of the formulas of the form $o(\bar{x}) \wedge \phi(\bar{x})$, then we can enumerate in CONSTANT-DELAY$_{lin}$ $\phi(\bar{x})$. In the sequel we can assume that there is a fixed ancestor-type $o(\bar{x})$ on the free variables of our MSO queries. Moreover, if the ancestor-type o requires variables x_i and x_j to be equal, we can replace all occurrences of x_j in the formula with x_i, enumerate the resulting formula and reconstruct on the fly the original solutions. Thus we may assume that the ancestor-type o ensures that all variables are distinct. Once $o(\bar{x})$ is fixed we say that two variables $x_i, x_j \in \bar{x}$ are *o-consecutive* if $o(\bar{x})$ implies that $x_i < x_j$ and no variable z of $\bar{x}$ is such that $x_i < z < x_j$.

3.4 It is enough to consider o-compatible queries.

Given an MSO query $\phi(\bar{x})$, an ancestor-type $o(\bar{x})$, and two o-consecutive variables $x_i < x_j$ of $\bar{x}$, we say that a formula $\alpha(x_i, x_j)$ *has its quantifications relativized to* $[x_i, x_j]$ if it is defined from arbitrary unary MSO queries and the descendant relation $<$, using first-order quantifications of the form $\exists y\ x_i \leq y \leq x_j \wedge \alpha$ or $\forall y\ x_i \leq y \leq x_j \rightarrow \alpha$ and, similarly, monadic second-order quantifications restricted to sets of elements within $[x_i, x_j]$. Note that such a formula may use *arbitrary unary* MSO subqueries. In particular they can test whether a node on the path from x_i to x_j is a left or right child. Moreover, they can test for an arbitrary MSO property of the subtree rooted at a sibling of any node on the path from x_i to x_j.

A formula $\phi(\bar{x})$ is said to be *o-compatible* if it is a conjunction of formulas $\alpha(x_i, x_j)$, where x_i, x_j are o-consecutive variables and α has its quantifications relativized to $[x_i, x_j]$. We say that two queries $\phi_1(\bar{x})$ and $\phi_2(\bar{x})$ are disjoint if for any binary tree T we have $\phi_1(T) \cap \phi_2(T) = \emptyset$.

The following result is a classical consequence of the Compositional Method developed by Shelah [Shelah 1975]. It essentially says that the MSO type of the tree can be derived from the MSO type of elements covering the tree. It can be proved using a simple Ehrenfeucht-Fraïssé game argument, see also Lemma 1 and Lemma 2 in [Colcombet 2007].

THEOREM 3.1. *Over binary trees, for every ancestor-type o and every* MSO *query $\phi(\bar{x}) \wedge o(\bar{x})$, there is an equivalent query which is a union of disjoint o-compatible queries.*

PROOF. Let k be the quantifier rank of ϕ. We start with some useful notation. Given a tree t, its MSO-k-type is the set of MSO sentences of quantifier rank k that hold in t. It is well known that there are only finitely many MSO-k-types and that each of them can be described by a sentence of MSO. Given a tree t and two nodes x, y of t, such that $x < y$, we denote by t_x the subtree of t rooted at x and by $t[x, y]$ the nodes of t that are descendants of x but not descendants of y. Given a tuple $\bar{a}$ of nodes of t we denote by $t(\bar{a})$ a recoloring of t such that the nodes of $\bar{a}$ have been distinguished.

A tuple such that $o(\bar{x})$ holds, induces a covering of the tree whose components are: t_{x_i} — if x_i has no o-consecutive variable, and $t[x_i, x_j]$ — if x_i and x_j are o-consecutive.

The Composition Method tells us that the MSO-k-types of each component taken separately induces the MSO-k-type of the whole tree. In other words, if we know the MSO-k-types of each of the t_{x_i} and $t[x_i, x_j]$, then we know the MSO-k-type of $t(\bar{x})$. This property is folklore and can be proved by a simple Ehrenfeucht-Fraïssé game argument. From the initial remarks, the MSO-k-type of t_{x_i} can be described by a unary MSO formula $\xi(x_i)$ and the MSO-k-type of $t[x_i, x_j]$ by an MSO formula $\xi(x_i, x_j)$. Hence the corresponding MSO-k-type of $t(\bar{x})$ is a conjunction of MSO formulas that are either unary or are binary and involve two o-consecutive variables.

Because the MSO-k-type of $t(\bar{x})$ implies whether $\phi(\bar{x})$ holds or not, $\phi(\bar{x})$ is a union of conjunctions as above. The union is finite because there are only finitely many MSO-k-types. The union is disjoint because each conjunct involves two different MSO-k-types for at least one component.

It remains to show that each formula $\xi(x_i, x_j)$ can be chosen with its quantifications relativized to $[x_i, x_j]$. This is exactly the Composition Theorem of Shelah [Shelah 1975]. The path from x_i to x_j is a linear order and we replace each subtree hanging off this path by its MSO-k-type. By this we mean that we recolor each node on the path from x_i to x_j by a color describing the MSO-k-type of the subtree hanging off that node. The result of Shelah says that $\xi(x_i, x_j)$ is equivalent to an MSO formula describing the resulting path, which has its quantifications relativized to $[x_i, x_j]$ by construction. The desired formula is then obtained by replacing the colors with the unary MSO

formulas describing the MSO-k-types of each removed subtree. □

As the o-compatible queries mentioned in Theorem 3.1 are disjoint, we can enumerate each of them separately. Hence it is enough to consider one o-compatible MSO query.

3.5 It is enough to consider o-compatible queries definable in $\Sigma_2(<)$.

This is the nontrivial step of our reductions. It exploits the following result of Colcombet based on a deterministic factorization forest theorem which can be seen here as an index structure for trees.

THEOREM 3.2 IMPLICIT IN [COLCOMBET 2007]. *Over binary trees, for every ancestor-type o and every o-compatible* MSO *query $\phi(\bar{x})$ there is an equivalent query which is still o-compatible but each of its conjuncts is of the form $\exists\bar{y}\forall\bar{z}\ \theta$, where θ is a disjunction of conjunctions of atomic predicates or* MSO *queries with one free variable or atoms using $<$.*

The result of Colcombet essentially shows that each of the conjuncts in o-compatible queries obtained from the decomposition lemma could be assumed to be definable with one alternation of first-order quantifiers, but using the ancestor relationship and unary MSO queries.

We denote by $\Sigma_2(<)$ the formulas of the form $\exists\bar{y}\forall\bar{z}\varphi$, where φ is a quantifier free formula using only atoms based on the relational predicates $<$ and P_a for all a ranging over a finite alphabet.

From Corollary 2.2 we know that each MSO query with one free variable can be evaluated in time linear in the size of the input tree. Therefore the unary MSO queries mentioned in Theorem 3.2 can be computed during the preprocessing phase, obtaining a tree over a new alphabet, where the new labels denote the old ones and whether each unary query holds or not at that node. We can therefore assume that our query is o-compatible and such that all its conjuncts are in $\Sigma_2(<)$. We denote by *o-simple $\Sigma_2(<)$ queries* these queries.

Note that the queries in $\Sigma_2(<)$ only have access to the labels and $<$. In particular they do not have access to the successor relation and don't distinguish between left and right children. This information is now part of the relabeling of the tree.

Altogether we have shown that it is enough to prove Theorem 2.3 for o-simple $\Sigma_2(<)$ queries over binary trees.

4. ENUMERATING SIMPLE $\Sigma_2(<)$ QUERIES.

From the previous section it follows that Theorem 2.3 is a consequence of the following:

PROPOSITION 4.1. *Over binary trees and for all o-simple $\Sigma_2(<)$ queries $\psi(\bar{x})$, Enum(ψ) belongs to* CONSTANT-DELAY$_{lin}$.

PROOF. The proof follows by an induction on the number of free variables in ψ. The base case (unary query) is done via our assumption that this variable denotes the root of the tree. So let us assume that we have the result for n-ary queries and we want to extend it to ones with $n+1$ variables.

The following proposition is the key ingredient for the inductive step:

PROPOSITION 4.2. *Given a binary tree T and a $\Sigma_2(<)$ formula $\phi(x,y)$ that logically implies $x<y$ and has its quantifications relativized to $[x,y]$ we can, in time $O(|T|)$, prepare an index structure that allows us to do the following: given a node u of T, enumerate all the solutions to $\phi(T)$ whose first component is u. Moreover, the enumeration procedure is done with constant delay.*

Before proving Proposition 4.2, we first conclude the proof of Proposition 4.1. Let $\psi(x_1,\ldots,x_{n+1})$ be an o-simple $\Sigma_2(<)$ query. Without loss of generality assume that x_n and x_{n+1} are o-consecutive variables with $x_n<x_{n+1}$ and, for all $i<n$, we don't have $x_{n+1}<x_i$ in o. By our choice of x_{n+1} and o-simplicity $\psi(x_1,\ldots,x_{n+1})$ must be of the form $\psi'(x_1,\ldots,x_n)\wedge\psi''(x_n,x_{n+1})$ where ψ'' is the $\Sigma_2(<)$ conjunct of ψ that describes the path from x_n to x_{n+1}. Let $\phi(x_1,\ldots,x_n)=\psi'(x_1,\ldots,x_n)\wedge\exists z\psi''(x_n,z)$. This formula is o'-simple where o' is the restriction of o to the first n variables. Therefore by induction hypothesis Enum(ϕ) is in CONSTANT-DELAY$_{lin}$. Let T be a binary tree. A CONSTANT-DELAY$_{lin}$ procedure for Enum(ψ) is:

The precomputation phase performs both the precomputation phase of Enum(ϕ) as given by the induction hypothesis, and the precomputation phase for ψ'' given by Proposition 4.2. This is done in $O(|T|)$ as required.

The enumeration phase nests the enumeration procedure for ψ'' given by Proposition 4.2 inside the enumeration procedure for $\phi(T)$ given by the induction hypothesis. Given a solution $(a_1, \ldots, a_n) \in \phi(T)$ we apply Proposition 4.2 to node a_n and find in constant delay all nodes a_{n+1} such that $(a_n, a_{n+1}) \in \psi''(T)$. Notice that, by definition of ϕ, we always have at least one such node a_{n+1} and that all such nodes are exactly those where the tuple $(a_1, \ldots, a_n, a_{n+1})$ is in $\psi(T)$. Therefore, for each such node a_{n+1} we output the tuple $(a_1, \ldots, a_n, a_{n+1})$. Once all the nodes a_{n+1} have been found, we continue the simulation of the enumeration procedure for $\phi(T)$ and obtain in constant delay the next tuple $(b_1, \ldots, b_n) \in \phi(T)$ and eventually compute all of $\psi(T)$. □

Proof of Proposition 4.2. Let T be a binary tree and $\phi(x, y)$ be a $\Sigma_2(<)$ formula that logically implies $x < y$ and has its quantifications relativized to $[x, y]$. We wish to enumerate $\phi(T)$. We start with some useful normalization of the formula.

Over words, a *monomial* is a language of the form $A_0^* a_1 A_1^* \ldots a_m A_m^*$, where each a_i is a fixed letter and A_i is a (possibly empty) set of letters. A *polynomial* is a finite union of monomials. In [Thomas 1982] a characterization of $\Sigma_2(<)$ over words was given that was later shown to be effective [Arfi 1987]:

Theorem 4.3 [Thomas 1982; Arfi 1987]. *Over words, a language is definable in $\Sigma_2(<)$ if and only if it is a polynomial.*

As we are dealing with formulas with free variables, we slightly extend the above definitions. Over words, a formula $\psi(x, y)$ is called a *monomial formula* if it holds for exactly those pairs (x, y), such that the subword between positions x and y (including both ends) matches a regular expression $a_0 A_0^* a_1 A_1^* \ldots a_m A_m^* a_{m+1}$, where each a_i is a fixed letter and A_i is a (possibly empty) set of letters. A *polynomial formula* is a finite disjunction of monomial formulas.

From Theorem 4.3 we immediately get

Corollary 4.4. *Over words, any $\Sigma_2(<)$ formula $\psi(x, y)$ that logically implies $x < y$ and has its quantifications relativized to $[x, y]$ is equivalent to a polynomial formula.*

In our case $\phi(x, y)$ is indeed a $\Sigma_2(<)$ formula, but it talks about trees. Fortunately, it also implies that $x < y$ and has its quantifications relativized to $[x, y]$, so for any two nodes $u < v$ of the input tree, $\phi(u, v)$ holds if and only if a word formed by the labels of the nodes on a path from u to v satisfies ϕ. Hence Corollary 4.4 implies:

Corollary 4.5. *Let $\phi(x, y)$ be a $\Sigma_2(<)$ formula that logically implies $x < y$ and has its quantifications relativized to $[x, y]$. Then there is a polynomial formula $\hat{\phi}$ such that for all trees T, $\phi(T) = \hat{\phi}(T)$.*

Let $\hat{\phi} = \bigvee_{j=1}^{j=k} \psi_j$ be the polynomial formula corresponding to ϕ as described by Corollary 4.5, where each ψ_j is a monomial formula. It is tempting at this point to enumerate separately each $\psi_j(T)$. Unfortunately, this will not fulfill the "no duplicate" constraint as a tuple may be present in $\psi_j(T)$ for several values of j. We will cope with this problem by considering all the monomial formulas at the same time using a product construction.

In order to jump quickly from one solution to another it will be enough to remember all the subparts of each of the monomial formulas that are currently satisfied at the current node. This is done as follows.

Fix for the moment an arbitrary $j \leq k$. As ψ_j is a monomial formula, it describes paths of the form $a_0 A_0^* a_1 A_1^* \ldots a_m A_m^* a_{m+1}$. Hence if $(u, v) \in \psi_j(T)$, the sequence of labels of the nodes on the path from u to v (denoted $[u, v]$) matches the regular expression $a_0 A_0^* a_1 A_1^* \ldots a_m A_m^* a_{m+1}$. We define the following suffixes of this regular expression: $e_0 = a_0 A_0^* a_1 A_1^* \ldots a_m A_m^* a_{m+1}$ and for each $1 \leq i \leq m+1$ let $e_i = A_{i-1}^* a_i A_i^* \ldots a_m A_m^* a_{m+1}$. Let also $\mathcal{S} = \{e_0, e_1, \ldots, e_{m+1}\}$. For two nodes $u < v$ of T, the j-type of (u, v) is exactly the set of regular expressions of $\mathcal{S}$ matched by $[u, v]$. We say that a j-type is *good* if it contains e_0. Clearly, $(u, v) \in \psi_j(T)$ if and only if the j-type of (u, v) is good.

We do this for all $j \leq k$ and define for two nodes $u < v$ of T the *type* of (u, v) as the tuple formed from all its j-types. Hence $(u, v) \in \phi(T)$ iff for some $1 \leq j \leq k$ its j-type is good. Notice that we have finitely many types and we call *good* those for which at least one component is good.

The following simple claim illustrates the key motivation for using types.

CLAIM 4.6. *Let T be a binary tree and $u < v < w, w'$ be nodes in T such that (u, w) has type τ_1 and both (v, w) and (v, w') have type τ_2. Then (u, w') has type τ_1.*

PROOF. As a type is a tuple containing all j-types, it is enough to show the claim for a fixed j-type. By symmetry it is enough to show that the j-type of (u, w) is included in the j-type of (u, w').

Assume that the j-type of (u, w) contains some suffix s. Hence there is a matching of s in the path from u to w, i.e. witnesses for the existentially quantified nodes. Fix such a matching α. α induces a matching of some suffix s' of s in the path from v to w. Since the j-types of (v, w) and (v, w') are the same, s' also matches the path from v to w'. Combining this later matching with α on the path from u to v (excluding v) provides a proper matching of s on the path from u to w'. □

In particular, in the scenario described in Claim 4.6, if $(u, w) \in \phi(T)$ (that is τ_1 is good), then $(u, w') \in \phi(T)$.

We say that a node u of a tree T is *valid for x* if there exists a node v such that $(u, v) \in \phi(T)$. Similarly a node v is *valid for y* if there exists a node u such that $(u, v) \in \phi(T)$. For a type τ and a node u we say that the pair (u, τ) is *interesting* if there exists a node v such that the type of (u, v) is τ and τ is good (in particular, u is valid for x). Note that all these properties are definable in MSO via unary queries, hence computable in time $O(|T|)$.

We now describe the index structure required by Proposition 4.2.

Definition 4.7. The *basic index structure* of T for $\phi(x, y)$ is the following directed graph:

Its vertices are the nodes of T valid for y and the interesting pairs of T.

Its edges are defined as follows:

• We have an edge between v and (u, τ) when the type of (u, v) is τ, and u is the "bottom-most" node of T with $(u, v) \in \phi(T)$ (i.e, $\forall w, u < w < v$ implies $(w, v) \notin \phi(T)$).

• We have an edge between (u, τ) and (u', τ') when $u' < u$ and there is a node $v > u$ with the type of (u, v) being τ and the type of (u', v) being τ' and u' is the "bottom-most" node with this property: for all nodes w with $u' < w < u$ there is no $v > u$ with the type of (u, v) being τ and $(w, v) \in \phi(T)$.

The key properties of this structure are summarized in the following lemma:

LEMMA 4.8. *The basic index structure of T for $\phi(x, y)$ has the following properties:*

(1) *It is computable in time $O(|T|)$.*

(2) *It is a forest with leaves being nodes valid for y and internal nodes being interesting pairs.*

(3) *For any node u valid for x and two different interesting pairs (u, τ) and (u, τ'), they occur in different trees inside the basic index structure.*

(4) *(*Completeness*) For all nodes u, v of the tree T, if $(u, v) \in \phi(T)$ and τ is the type of (u, v), then v is a leaf in the subtree of (u, τ) inside the basic index structure.*

(5) *(*Soundness*) For all nodes u, v of the tree T, if τ is the type of (u, v) and v is a leaf in the subtree of (u, τ) inside the basic index structure, then $(u, v) \in \phi(T)$.*

PROOF. We start with Property (1).

From the remark above, the nodes of the basic index structure are definable in MSO and therefore they can be computed in time linear in $|T|$ using Corollary 2.2. For the first kind of edges notice that each v uniquely determines a u and that the relation between u and v can also be described in MSO. Similarly, for the second kind of edges, each interesting pair (u, τ) uniquely determines a u' and their relationship can be described in MSO. Hence Corollary 2.2 can be invoked again to compute the edges in time $O(|T|)$.

We continue with Property (2).

Clearly the structure is acyclic as a node can only point to an ancestor in T. Moreover, each node v valid for y has a u such that $(u, v) \in \phi(T)$ and hence nodes of this kind cannot be internal nodes and have a unique parent, corresponding to the bottom-most u such that $(u, v) \in \phi(T)$.

Similarly, an interesting pair (u, τ) either has no parent or a parent of the form (u', τ'). Recall from the previous point that u' is uniquely determined by (u, τ). It remains to show that τ' is also uniquely determined by (u, τ). Assume this is not the case and that (u, τ) is associated to both (u', τ'_1) and (u', τ'_2). Then, by construction, there exist v_1, v_2 such that τ is the type of both (u, v_1) and (u, v_2), τ'_1 is the type of (u', v_1) and τ'_2 the type of (u', v_2). Claim 4.6 implies that $\tau'_1 = \tau'_2$ and therefore our basic index structure is a forest.

Property (3) is immediate as we already know that the basic index structure is in fact a forest and that a necessary condition for (u', τ') to be a parent of (u, τ) is that $u' < u$.

We now move to Property (4). Assume $(u, v) \in \phi(T)$, τ is the type of (u, v) and that $u_0 = u < u_1 < \ldots < u_t < v$ are all the nodes of T on the path from u to v such that for each $0 \leq i \leq t$ the pair $(u_i, v) \in \phi(T)$. Let τ_i denote the type of (u_i, v). From the construction of the basic index structure we know that v is a child of (u_t, τ_t).

We show that for all $0 < i \leq t$, (u_i, τ_i) is the child of (u_{i-1}, τ_{i-1}) in the basic index structure. If this was not the case, then there exist nodes w, v', with $u_{i-1} < w < u_i < v'$, the type of (u_i, v') being τ_i and $(w, v') \in \phi(T)$. As τ_i is also the type of (u_i, v), Claim 4.6 implies that the type of (w, v) is the same as the type of (w, v') and therefore $(w, v) \in \phi(T)$, a contradiction.

It remains to show Property (5). We show that if v is in the subtree of (u, τ) inside the basic index structure, then the type of (u, v) is τ. As (u, τ) is interesting pair, τ is (by definition) good and thus $(u, v) \in \phi(T)$. The proof is a simple induction on the distance between v and (u, τ) inside the basic index structure. If (u, τ) is a parent of v, then the type of (u, v) is τ by the definition of the parent relation of leaves. The inductive step is again a direct consequence of Claim 4.6. □

Lemma 4.8 (in particular properties (4) and (5)) justifies that the basic index structure is an adequate tool for the enumeration of the solutions to ϕ. In order to obtain the desired constant delay procedure we need to extend this structure a bit for navigating efficiently within the basic index structure. The resulting structure is called the *full index structure of T for ϕ*. It contains the following enhancements.

Recall from Lemma 4.8, Property (1), that the basic index structure can be computed in time $O(|T|)$ and has therefore a size linear in $|T|$.

By Property (2) the basic index structure is a forest. It will be important that we have access to the descendant relation inside this forest in constant time. To do this we perform a depth-first traversal of the underlying forest and compute a *dfs number* to each node (corresponding to the last time we have visited it). Clearly, every node has a number larger than all the nodes in its subtree and these numbers are computed in time $O(|T|)$.

We enrich furthermore the basic index structure by associating to each leaf the next leaf in the dfs traversal. This will allow us to jump from one leaf to the next one in constant time. Moreover, to each internal node (u, τ) we add an additional pointer to the first leaf of its subtree. Note that this can be easily achieved in time $O(|T|)$.

Finally, for each node u of the original input tree that is valid for x, we keep a pointer to each of the interesting pairs (u, τ) inside the full index structure. This can be done in time $O(|T|)$ as every such node u requires a number of pointers bounded by the number of different types, which does not depend on T.

This completes the construction of the index structure for Proposition 4.2 and the precomputation phase. Note that it follows from the description above that this phase is achieved in time $O(|T|)$.

We now turn to the enumeration phase which, as expected, is a simple traversal of our index structure. Let u be a node in T which is valid for x. We wish to enumerate all v such that $(u, v) \in \phi(T)$. To do this we consider in turn all the interesting pairs (u, τ) using the appropriate precomputed pointers. For each such τ we jump, in constant time using the precomputed pointer,

to the first leaf v in the subtree of (u, τ) in the full index structure and output (u, v). Now, as long as we remain in the subtree of (u, τ) (this can easily be checked in constant time by comparing the dfs numbers) we successively go through all the leaves of our index structure in dfs order (again in constant time, using the precomputed pointers) outputting the corresponding pair (u, v).

Clearly, between outputting consecutive solutions there is only a constant delay and each pair we output is unique by Property (3). From Property (5) of Lemma 4.8 we know that each pair that we output is in $\phi(T)$. Property (4) ensures that we do not skip any solution.

This completes the proof of Proposition 4.2.

5. CONCLUSION

As mentioned in the introduction, Theorem 2.3 immediately lifts to structures of bounded tree-width: for all queries $\phi \in$ MSO and all $k \in \mathbb{N}$ there is a linear time and constant delay algorithm enumerating the output of ϕ over any structure of tree-width k. The reason is that any graph of tree-width k can be interpreted over a tree via an MSO query and Theorem 2.3 can be applied on the composition of ϕ and the formula providing the interpretation. The details can be found in [Bagan 2006].

It should be noted that our enumeration algorithm, as well as the one of Bagan, are non elementary in the size of the formula. This cannot be avoided unless P=NP [Frick and Grohe 2004]. In our case the non elementary blow-up is hidden in Theorem 2.1, the Ehrenfeucht-Fraïssé argument of Theorem 3.1 and in the result of Colcombet, Theorem 3.2.

Our notion of linear time and constant delay requires furthermore that the total extra memory used during the enumeration phase is constant. It is not clear that the enumeration algorithm of Bagan has this extra property as it uses nested subprocess pushing (in constant time) pointers to the parent process on a stack. The nesting depth of this subprocess seems to be arbitrary.

However the result of Bagan also deals with MSO formulas containing free monadic second-order variables. In this case the delay is linear in the size of the solution being output. This cannot be avoided as the algorithm needs at least the time to output the solution. It is not clear how our technique can be lifted in order to take care of monadic second-order variables.

REFERENCES

Arfi, M. 1987. Polynomial Operations on Rational Languages. In *Symp. on Theoretical Aspects of Computer Science (STACS)*. 198–206.

Bagan, G. 2006. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Conf. on Computer Science Logic (CSL)*. 167–181.

Colcombet, T. 2007. A Combinatorial Theorem for Trees. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*. 901–912.

Courcelle, B. 1990. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. 193–242.

Courcelle, B. 2009. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics 157,* 12, 2675–2700.

Durand, A. and Grandjean, E. 2007. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. on Computational Logic (ToCL) 8,* 4.

Flum, J., Frick, M., and Grohe, M. 2002. Query evaluation via tree-decompositions. *J. of the ACM 49,* 6, 716–752.

Frick, M. and Grohe, M. 2004. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic 130,* 1-3, 3–31.

Kazana, W. and Segoufin, L. 2011. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science (LMCS) 7,* 2.

Seese, D. 1996. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science 6,* 6, 505–526.

Shelah, S. 1975. The monadic theory of order. *Annals of Mathematics 102,* 3, 379–419.

Thomas, W. 1982. Classifying Regular Events in Symbolic Logic. *J. on Computer and System Sciences (JCSS) 25,* 3, 360–376.

...